合肥工業大學

信息安全实验报告

密码学实验

姓　　名 袁焕发

学　　号 2019217769

专业班级 物联网工程 19-2

指导教师 周健

院系名称 计算机与信息学院

2022　年　 04　月　11　日

一、实验目的

实现 AES、DES、RSA 算法，完成加密解密操作。

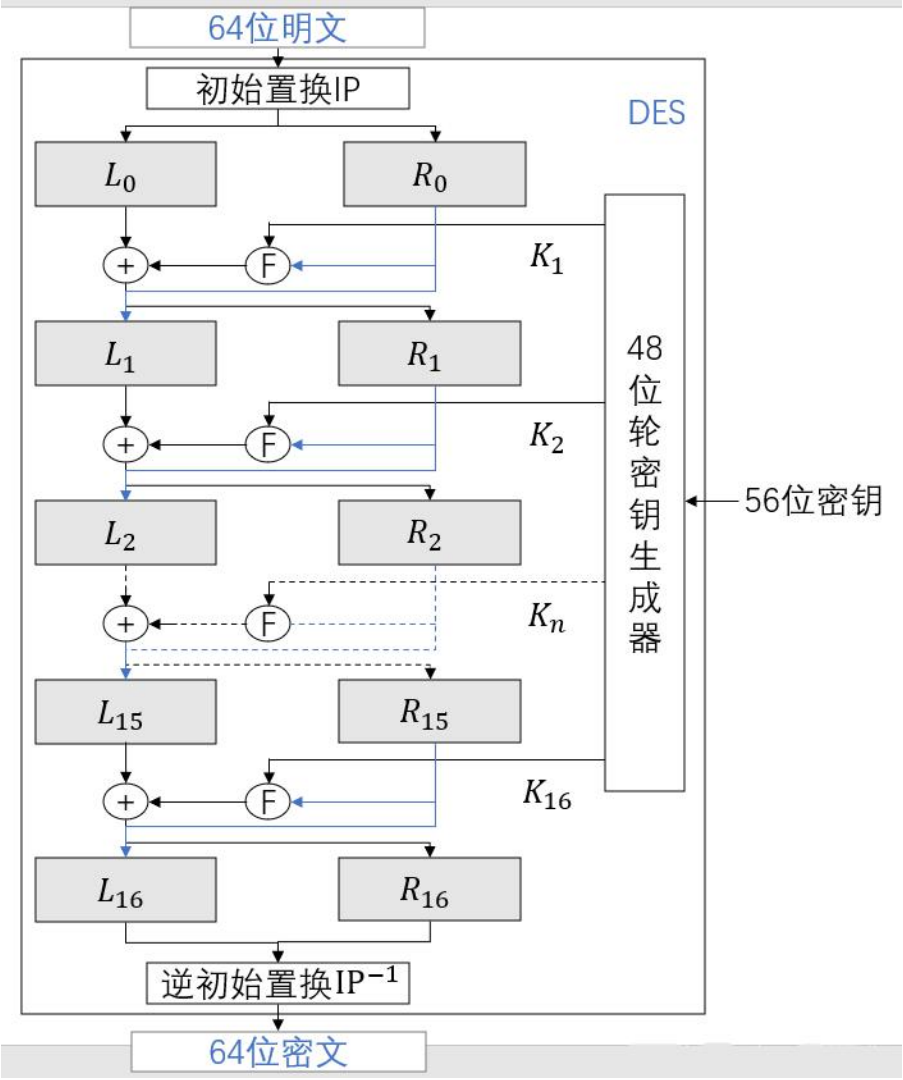
二、实验环境

Windows 10、IntelliJ IDEA 2021.3.3、Java1.7 JDK

三、实验原理

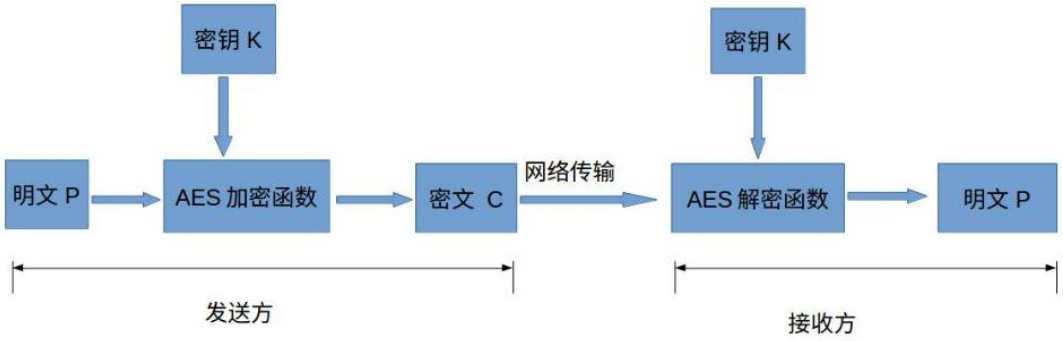
1. DES 加密

DES 算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是 1972 年美国 IBM 公司研制的对称密码体制加密算法。明文按 64 位进行分组，密钥长 64 位，密钥事实上是 56 位参与 DES 运算（第 8、16、24、32、40、48、56、64 位是校验位，使得每个密钥都有奇数个 1），分组后的明文组和 56 位的密钥按位替代或交换的方法形成密文组的加密方法。



2. AES 加密

AES 全称为 Advanced Encryption Standard，是美国联邦政府采用的一种区块加密标准，用来替代原先的 DES。



AES 算法在对明文加密的时候，并不是把整个明文一股脑加密成一整段密文，而是把明文拆分成一个个独立的明文块，每一个明文块长度 128bit。这些明文块经过 AES 加密器的复杂处理，生成一个个独立的密文块，这些密文块拼接在一起，就是最终的 AES 加密结果

设 AES 加密函数为 E，则 C = E(K, P),其中 P 为明文，K 为密钥，C 为密文。也就是说，把明文 P 和密钥 K 作为加密函数的参数输入，则加密函数 E 会输出密文 C。设 AES 解密函数为 D，则 P = D(K, C),其中 C 为密文，K 为密钥，P 为明文。也就是说，把密文 C 和密钥 K 作为解密函数的参数输入，则解密函数会输出明文 P。

3．RSA 加密

RSA 加密算法，是世界上第一个非对称加密算法，也是数论的第一个实际应用。它的算法如下：

1.找两个非常大的质数 p 和 q（通常 p 和 q 都有 155 十进制位或都有 512 十进制位）并计算 n=pq，k=(p-1)(q-1)。

2.将明文编码成整数 M，保证 M 不小于 0 但是小于 n。

3.任取一个整数 e，保证 e 和 k 互质，而且 e 不小于 0 但是小于 k。加密钥匙（称作公钥）是(e,n)。

4.找到一个整数 d，使得 ed 除以 k 的余数是 1（只要 e 和 n 满足上面条件，d 肯定存在）。解密钥匙（称作密钥）是(d,n)。

加密过程：加密后的编码 C 等于 M 的 e 次方除以 n 所得的余数。

解密过程：解密后的编码 N 等于 C 的 d 次方除以 n 所得的余数。

只要 e、d 和 n 满足上面给定的条件。M 等于 N。

四、实验步骤

1．DES 加密

主要模块

```java
 * 密文分组，密文一定是64位的倍数，分组时不需要补齐数据
 */
public static String[] divide4Decrypt(String cipher) {
    String[] cipherGroup = new String[cipher.length() >>> 6];
    for (int i = 0; i < cipher.length(); i += 64) {
        cipherGroup[i >>> 6] = cipher.substring(i, i + 64);
    }
    return cipherGroup;
}
```

构建明文密文分组

```java
public static String feistel(String[] plainTextGroup, String[] ks) {
    StringBuilder cipher = new StringBuilder( capacity: plainTextGroup.length << 64);
    for (int i = 0; i < plainTextGroup.length; i++) {
        // 2. 初始转置
        String lr = transpose(plainTextGroup[i], BEGIN); // 某一组明文
        // 3. 16轮加密
        String l = lr.substring(0, 32); //
        String r = lr.substring(32);
        String li = l;
        String ri = r;
        for (int j = 0; j < ks.length; j++) {
            // 3.1 将lr右半部分(32位)扩展为48位
            String rr = transpose(ri, E);
            // 3.2 将ri与ki异或运算得到48位数据
            String rk = xor(rr, ks[j]);
            // 3.3 查S盒
            String rrr = searchSBox(rk);
            // 3.4 转置得到RR(32位)
            rrr = transpose(rrr, P);
            // 3.5 rrr与li异或得到新的ri，原来的ri赋值给li
            String newRi = xor(rrr, li);
            li = ri;
            ri = newRi;
        }
        // li - r15 ri r16
        // 4. 求逆转置，得到这一组对应的密文
        String result = transpose( source: ri + li, END); // 这一组加密的结果
        cipher.append(result);
    }
```

调用 S 盒函数，进行十六轮加密置换，再逆置，得到加密结果。

测试结果如下

```
D:\Java\JDK17\bin\java.exe "-javaagent:D:\IntelliJ IDEA\lib\idea_rt.jar=54218:D:\IntelliJ IDEA\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\yuanhuanfa
DES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
1
请输入明文(不含中文)：2019217769
请输入密钥(8个字符以内，不含中文)：123456
加密结果：1100001011100100100100011001100110101010110111010001110010100000000101001001110011111000000010011110110000111011001110001010011
DES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
2
请输入密文(不含中文)：1100001011100100100100011001100110101010110111010001110010100000000101001001110011111000000010011110110000111011001110001010011
请输入密钥(8个字符以内，不含中文)：123456
解密结果：2019217769
DES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
```

2．AES 加密

明文、密钥分组实现模块

```
/**
 * 1. 将明文的普通字符串转化成对应的多个字节数组
 * 2. 每个字节数组都不能超过16字节
 * 3. 将字节数组按列排成4 * 4矩阵
 * 4. 字节不足时用00充当
 */
public static String[][][] divide4PlainText(String plainText
    byte[] bytes = plainText.getBytes();
    int len = (bytes.length & 15) == 0  ? (bytes.length >>>
    String[][][] ret = new String[len][4][4];
    for (int k = 0; k < len; k++) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                int index = (i << 2) + j + (k << 4);
                if (index < bytes.length) {
                    String hex = Integer.toHexString(bytes[i
                    if (hex.length() == 1) {
                        hex = "0" + hex;
                    }
                    ret[k][j][i] = hex;
                } else {
                    ret[k][j][i] = "00";
                }
            }
        }
    }
    return ret;
}
```

```
*/
public static String[][] divide4Secret(String secret) {
    String[][] ret = new String[4][4];
    byte[] bytes = secret.getBytes();
    if (bytes.length > 16) {
        throw new RuntimeException("密钥不能超过16个字节");
    }
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            int index = (i << 2) + j;
            if (index < bytes.length) {
                String hex = Integer.toHexString(bytes[index]);
                if (hex.length() == 1) {
                    hex = "0" + hex;
                }
                ret[j][i] = hex;
            } else {
                ret[j][i] = "00";

            }
        }
    }
    return ret;
}
```

实现列混淆

```
/**
 * 列混淆
 */
public static String[][] fixColumn(String[][] source, String[][] fix) {
    String[][] result = new String[4][4];
    for (int i = 0; i < 4; i++) {   // 控制行
        for (int j = 0; j < 4; j++) {   // 控制行使用次数
            String ret = "00000000";
            for (int k = 0; k < 4; k++) {   // 控制列
                String binMultiply = binMultiply(fix[i][k], source[k][j]);
                ret = xor(ret, binMultiply);
            }
            // 将结果转为16进制
            String hex = binToHex(ret);
            result[i][j] = hex;
        }
    }
    return result;
}
```

加密、解密函数

```
// 1. 构造密钥组
String[][][] extendSecret = extendSecret(initSecret);
for (int i = 0; i < plainMatrix.length; i++) {
    // 2. 轮密钥加
    plainMatrix[i] = xor4HexArr(plainMatrix[i], extendSecret[0]);
    // 3.10轮重复加密操作
    for (int round = 1; round <= 10; round++) {
        // 字节代替
        replaceByte(plainMatrix[i], S);
        // 行移位
        plainMatrix[i] = moveRow(plainMatrix[i]);
        // 列混淆(第十轮不进行)
        if (round != 10) {
            plainMatrix[i] = fixColumn(plainMatrix[i], FIX_COLUMN);
        }
        // 轮密钥加
        plainMatrix[i] = xor4HexArr(plainMatrix[i], extendSecret[rou
    }
    // 4.拼接密文
```

```
// 0. 密文/密钥分组
String[][][] cipherMatrix = divide4Hex(cipher);
String[][] initSecret = divide4Secret(secret);
List<Byte> list = new ArrayList<>( initialCapacity: cipherMatrix.length << 4 + 16);
// 1. 构造密钥组
String[][][] extendSecret = extendSecret(initSecret);
for (int i = 0; i < cipherMatrix.length; i++) {
    // 2. 轮密钥加
    cipherMatrix[i] = xor4HexArr(cipherMatrix[i], extendSecret[10]);
    // 3.10轮重复加密操作
    for (int round = 9; round >= 0; round--) {
        // 逆行移位
        cipherMatrix[i] = moveRowInverse(cipherMatrix[i]);
        // 逆字节代替
        replaceByte(cipherMatrix[i], S_INVERSE);
        // 轮密钥加
        cipherMatrix[i] = xor4HexArr(cipherMatrix[i], extendSecret[round]);
        // 逆列混淆(第十轮不进行)
        if (round != 0) {
            cipherMatrix[i] = fixColumn(cipherMatrix[i], FIX_COLUMN_INVERSE);
        }
    }
    // 4.还原明文
    for (int k = 0; k < 4; k++) {
```

测试结果如下

```
D:\Java\JDK17\bin\java.exe "-javaagent:D:\IntelliJ IDEA\lib\idea_rt.jar
AES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
1
请输入明文(不含中文)：2019217769
请输入密钥(16个字符以内，不含中文)：00000000
加密结果：92e6e3d9f0b5c46c231e2a22fbaca174
AES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
2
请输入密文(不含中文)：92e6e3d9f0b5c46c231e2a22fbaca174
请输入密钥(16个字符以内，不含中文)：00000000
解密结果：2019217769
AES算法，请选择您要进行的操作(按下对应操作数字后回车)：(1)加密 (2)解密 (3)退出
```

3．RSA 加密

调用函数生成 RSA 加密所需的 e n d p q。

```
 */
public String[] createKey(int keylen) {// 输入密钥长度
    String[] output = new String[5]; // 用来存储密钥的e n d p q
    try {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(keylen); // 指定密钥的长度，初始化密钥对生成器
        KeyPair kp = kpg.generateKeyPair(); // 生成密钥对
        RSAPublicKey puk = (RSAPublicKey) kp.getPublic();
        RSAPrivateCrtKey prk = (RSAPrivateCrtKey) kp.getPrivate();
        BigInteger e = puk.getPublicExponent();
        BigInteger n = puk.getModulus();
        BigInteger d = prk.getPrivateExponent();
        BigInteger p = prk.getPrimeP();
        BigInteger q = prk.getPrimeQ();
        output[0] = e.toString();
        output[1] = n.toString();
        output[2] = d.toString();
        output[3] = p.toString();
        output[4] = q.toString();
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, msg: null, ex);
    }
    //System.out.println(output[0]);
    return output;
}
```

加密和解密模块

```java
public String encrypt(String clearText, String eStr, String nStr) {
    String secretText = "";

    try {
        clearText = URLEncoder.encode(clearText, enc: "GBK");
        byte[] text = clearText.getBytes( charsetName: "GBK");//将字符串转换成by
        BigInteger mm = new BigInteger(text);//二进制串转换为一个大整数
        clearText = mm.toString();

        BigInteger e = new BigInteger(eStr);
        BigInteger n = new BigInteger(nStr);
        byte[] ptext = clearText.getBytes(StandardCharsets.UTF_8); // 获取明
        BigInteger m = new BigInteger(ptext);
        BigInteger c = m.modPow(e, n);
        secretText = c.toString();
    } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, msg: null,
    }
    return secretText;
}
```

```java
ublic String decrypt(String secretText, String dStr, String nStr)
    StringBuilder clearTextBuffer = new StringBuilder();
    BigInteger d = new BigInteger(dStr);// 获取私钥的参数d,n
    BigInteger n = new BigInteger(nStr);
    BigInteger c = new BigInteger(secretText);
    BigInteger m = c.modPow(d, n);// 解密明文
    byte[] mt = m.toByteArray();// 计算明文对应的字符串并输出
    for (int i = 0; i < mt.length; i++) {
        clearTextBuffer.append((char) mt[i]);
    }
    String temp = clearTextBuffer.toString();//temp为明文的字符串形式
    BigInteger b = new BigInteger(temp);//b为明文的BigInteger类型
    byte[] mt1 = b.toByteArray();

    try {
        String clearText = (new String(mt1, charsetName: "GBK"));
        clearText = URLDecoder.decode(clearText, enc: "GBK");
        return clearText;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }

    return null;
```

测试结果如下

```
D:\Java\JDK17\bin\java.exe "-javaagent:D:\IntelliJ IDEA\lib\idea_rt.jar=54749:D:\IntelliJ IDEA\bin" -Dfile.enc
key :
65537
key :
132201925785456391841776848599612324009325434178623536836898918591516678367128396758274449018054863637435475151
key :
481595381616833337114019681931137502329559338181514553012449991062677026410923102552846089995371971822513409907
key :
112478269684066217264316622768438025154448819210151420925001236546687889080620304675450663316691001553855613666
key :
117535525890281584918777606198555958477771433357159658893433094520558247955191202933274207028639398593643148205
5
明文是：2019217769测试
密文是：80343253638024000627799567412486995058676313205384034038135318633139314363883079502569586673426345526038
解密后的明文是：2019217769测试
```

输出结果的 key 依次为 e n d p q。

五、总结

通过本次实验，加深了我对 DES、AES、RSA 算法的原理的进一步认识。了解了算法中功能的代码实现步骤，完成了利用程序完成加密算法的过程。

六、附：源代码

1．DES

```java
package DESshiyan;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;

import java.util.List;
import java.util.Scanner;
```

```
public class DES {

    public static final int[] BEGIN = new int[] { 58, 50, 42, 34, 26, 18, 10, 2, 60,
52, 44, 36, 28, 20, 12, 4, 62, 54,
            46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25,
17, 9, 1, 59, 51, 43, 35, 27, 19,
            11, 3, 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 }; // 64
位
    public static final int[] END = new int[] { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7,
47, 15, 55, 23, 63, 31, 38, 6, 46,
            14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20,
60, 28, 35, 3, 43, 11, 51, 19, 59,
            27, 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 }; // 64 位

    public static final int[] PC_1 = new int[] { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50,
42, 34, 26, 18, 10, 2, 59, 51,
            43, 35, 27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7, 62,
54, 46, 38, 30, 22, 14, 6, 61, 53,
            45, 37, 29, 21, 13, 5, 28, 20, 12, 4 };

    public static final int[] PC_2 = new int[] { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21,
10, 23, 19, 12, 4, 26, 8, 16,
            7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49,
39, 56, 34, 53, 46, 42, 50, 36,
            29, 32 };

    public static final int[] E = new int[] { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 6, 9, 10,
11, 12, 13, 12, 13, 14, 15,
            16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28,
29, 28, 29, 30, 31, 32, 1 };

    public static final int[] P = new int[] { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,
26, 5, 18, 31, 10, 2, 8, 24, 14,
            32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    public static final int[][] S_BOX = new int[][] {
            { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7, 0, 15, 7, 4, 14, 2, 13,
1, 10, 6, 12, 11, 9, 5, 3,
                    8, 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0, 15, 12, 8,
2, 4, 9, 1, 7, 5, 11, 3, 14, 10,
                    0, 6, 13 }, // s1
            { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10, 3, 13, 4, 7, 15, 2, 8,
14, 12, 0, 1, 10, 6, 9, 11,
                    5, 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15, 13, 8, 10,
1, 3, 15, 4, 2, 11, 6, 7, 12, 0,
                    5, 14, 9 }, // s2
```

```
            { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8, 3, 13, 4, 7, 15, 2, 8,
14, 12, 0, 1, 10, 6, 9, 11,
                    5, 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15, 13, 8, 10,
1, 3, 15, 4, 2, 11, 6, 7, 12, 0,
                    5, 14, 9 }, // s3
            { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15, 13, 8, 11, 5, 6, 15, 0,
3, 4, 7, 2, 12, 1, 10, 14,
                    9, 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4, 3, 15, 0, 6,
10, 1, 13, 8, 9, 4, 5, 11, 12,
                    7, 2, 14 }, // s4
            { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9, 14, 11, 2, 12, 4, 7, 13,
1, 5, 0, 15, 10, 3, 9, 8,
                    6, 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14, 11, 8, 12,
7, 1, 14, 2, 13, 6, 15, 0, 9,
                    10, 4, 5, 3, }, // s5
            { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11, 10, 15, 4, 2, 7, 12, 9,
5, 6, 1, 13, 14, 0, 11, 3,
                    8, 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6, 4, 3, 2, 12,
9, 5, 15, 10, 11, 14, 1, 7, 6,
                    0, 8, 13 }, // s6
            { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1, 13, 0, 11, 7, 4, 9, 1,
10, 14, 3, 5, 12, 2, 15, 8,
                    6, 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2, 6, 11, 13,
8, 1, 4, 10, 7, 9, 5, 0, 15, 14,
                    2, 3, 12 }, // s7
            { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7, 1, 15, 13, 8, 10, 3, 7,
4, 12, 5, 6, 11, 0, 14, 9,
                    2, 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8, 2, 1, 14, 7,
4, 10, 8, 13, 15, 12, 9, 0, 3,
                    5, 6, 11, } // s8
    };

    public static final int[] K_MOVE = new int[] { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2,
2, 2, 1 };
    public static final String ZEROS = "00000000";

    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
         String choose = null;
         while (true) {
                System.out.println("DES 算法，请选择您要进行的操作(按下对应
操作数字后回车): (1)加密  (2)解密  (3)退出");
                choose = scanner.nextLine();
                if ("1".equals(choose)) {
                    System.out.print("请输入明文(不含中文): ");
                    String plainText = scanner.nextLine();
```

```
                System.out.print("请输入密钥(8 个字符以内，不含中文): ");
                String secret = scanner.nextLine();
                String cipher = encrypt(plainText, secret);
                System.out.println("加密结果: " + cipher);
            } else if ("2".equals(choose)) {
                System.out.print("请输入密文(不含中文): ");
                String cipher = scanner.nextLine();
                System.out.print("请输入密钥(8 个字符以内，不含中文): ");
                String secret = scanner.nextLine();
                String plainText = decrypt(cipher, secret);
                System.out.println("解密结果: " + plainText);
            } else if ("3".equals(choose)) {
                System.exit(0);
            }
        }

    }

    public static String encrypt(String plainText, String secret) throws
UnsupportedEncodingException {
        // 1. 构造密钥
        String[] ks = buildKs(secret);
        // 2. 明文分组
        String[] divide = divide(plainText);
        // 3. 调用 Feistel 函数
        return feistel(divide, ks);
    }

    public static String decrypt(String cipher, String secret) throws
UnsupportedEncodingException {
        // 1. 构造密钥, 解密时密钥倒转
        String[] ks = buildKs(secret);
        String[] decryptKs = new String[ks.length];
        for (int i = 0; i < ks.length; i++) {
            decryptKs[i] = ks[ks.length - i - 1];
        }
        // 2. 密文分组
        String[] cipherGroup = divide4Decrypt(cipher);
        // 3. 调用 Feistel 函数
        String plainText = feistel(cipherGroup, decryptKs);
        // 4. 将二进制明文转为原来的字符串
        List<Byte> byteList = new ArrayList<>();
        for (int i = 0; i < plainText.length(); i += 8) {
            String bitStr = plainText.substring(i, i + 8);
            if (!ZEROS.equals(bitStr)) {
                byte byteVal = Integer.valueOf(bitStr, 2).byteValue();
```

```
                byteList.add(byteVal);
            }
        }
        byte[] bytes = new byte[byteList.size()];
        for (int i = 0; i < byteList.size(); i++) {
            bytes[i] = byteList.get(i);
        }
        return new String(bytes, StandardCharsets.UTF_8);
    }

    public static String feistel(String[] plainTextGroup, String[] ks) {
        StringBuilder cipher = new StringBuilder(plainTextGroup.length << 64);
        for (int i = 0; i < plainTextGroup.length; i++) {
            // 2. 初始转置
            String lr = transpose(plainTextGroup[i], BEGIN); // 某一组明文
            // 3. 16 轮加密
            String l = lr.substring(0, 32); //
            String r = lr.substring(32);
            String li = l;
            String ri = r;
            for (int j = 0; j < ks.length; j++) {
                // 3.1 将 lr 右半部分(32 位)扩展为 48 位
                String rr = transpose(ri, E);
                // 3.2 将 ri 与 ki 异或运算得到 48 位数据
                String rk = xor(rr, ks[j]);
                // 3.3 查 S 盒
                String rrr = searchSBox(rk);
                // 3.4 转置得到 RR(32 位)
                rrr = transpose(rrr, P);
                // 3.5 rrr 与 li 异或得到新的 ri，原来的 ri 赋值给 li
                String newRi = xor(rrr, li);
                li = ri;
                ri = newRi;
            }
            // li - r15 ri r16
            // 4. 求逆转置，得到这一组对应的密文
            String result = transpose(ri + li, END); // 这一组加密的结果
            cipher.append(result);
        }
        return cipher.toString();
    }

    /**
     * 将明文按 64 位分组
     * 明文转为二进制可能不是 64 数的倍数，需要在分组后补 0
     *
```

```
     * @throws UnsupportedEncodingException
     */
    public static String[] divide(String text) throws
UnsupportedEncodingException {
        String zeros = "00000000";
        StringBuilder sb = new StringBuilder(text.length() << 3);
        // 将每一个字符转成二进制，拼接
        byte[] bytes = text.getBytes(StandardCharsets.UTF_8);
        for (int i = 0; i < bytes.length; i++) {
            String bitStr = Integer.toBinaryString(bytes[i]);
            int len = 8 - bitStr.length(); // 前面补 0
            if (len > 0) {
                bitStr = zeros.substring(0, len) + bitStr;
            }
            sb.append(bitStr);
        }
        // 若二进制不是 64 位的倍数，后面补 0
        int num = sb.length() & (1 << 6) - 1;
        while (num != 64) {
            sb.append(zeros);
            num += 8;
        }
        // 分组
        String[] result = new String[sb.length() >>> 6];
        for (int i = 0; i < sb.length(); i += 64) {
            result[i >>> 6] = sb.substring(i, i + 64);
        }
        return result;
    }

    /**
     * 密文分组，密文一定是 64 位的倍数，分组时不需要补齐数据
     */
    public static String[] divide4Decrypt(String cipher) {
        String[] cipherGroup = new String[cipher.length() >>> 6];
        for (int i = 0; i < cipher.length(); i += 64) {
            cipherGroup[i >>> 6] = cipher.substring(i, i + 64);
        }
        return cipherGroup;
    }

    /**
     * 将二进制数按照某数组进行置换
     */
    public static String transpose(String source, int[] reverseArr) {
        /*
```

```
            * (下标从 0 开始)取出 source 的第 reverseArr[i] - 1 位，作为转置结果的第 i 位
            */
            StringBuilder sb = new StringBuilder(source.length() + 16);
            for (int i = 0; i < reverseArr.length; i++) {
                sb.append(source.charAt(reverseArr[i] - 1));
            }
            return sb.toString();
        }

        /**
         * 构造秘钥组(16 个 48 位 ki)
         *
         * @param secret
         * @return
         */
        public static String[] buildKs(String secret) {
            // 构造 64 位初始秘钥
            StringBuilder sb = new StringBuilder(1 << 6);
            byte[] bytes = secret.getBytes();
            if (bytes.length > 8) {
                throw new RuntimeException("秘钥必须在 8 个字节以内");
            }
            for (int i = 0; i < bytes.length; i++) {
                String bitStr = Integer.toBinaryString(bytes[i]);
                int len = 8 - bitStr.length(); // 前面补 0
                if (len > 0) {
                    bitStr = ZEROS.substring(0, len) + bitStr;
                }
                sb.append(bitStr);
            }
            // 补齐 64 位
            while (sb.length() <= 64) {
                sb.append(ZEROS);
            }
            // 置换，得到 56 位有效秘钥
            String validSecret = transpose(sb.toString(), PC_1);

            // 分别对前后 28 位进行左移调度
            String[] ks = new String[16];
            String ci = validSecret.substring(0, 28);
            String di = validSecret.substring(28, 56);
            for (int i = 0; i < K_MOVE.length; i++) {
                int j = K_MOVE[i];
                ci = ci.substring(j) + ci.substring(0, j);
                di = di.substring(j) + di.substring(0, j);
```

```
            // ci + di => 置换 => ki
            ks[i] = transpose(ci + di, PC_2);
        }
        return ks;
    }

    /**
     * 根据右半部分 ri 和秘钥 ki 异或后的加密数据(48 位)查询 S 盒
     *
     * @param rk
     * @return 返回 32 位加密数据
     */
    public static String searchSBox(String rk) {
        StringBuilder sb = new StringBuilder(32);
        for (int i = 0; i < rk.length(); i += 6) {
            // (1)获取 6 位一组的数据
            String s = rk.substring(i, i + 6);
            // (2)取出首尾两位计算出行，取出中间四位，计算出列
            int row = Integer.valueOf("" + s.charAt(0) + s.charAt(5), 2);
            int col = Integer.valueOf(s.substring(1, 5), 2);
            // (3)根据索引获取到 S 盒中对应的值，转成 4 位二进制数据
            int index = ((row << 4) + col);
            s = Integer.toBinaryString(S_BOX[i / 6][index]);
            // 补齐四位
            int len = s.length();
            for (int j = 0; j < 4 - len; j++) {
                s = "0" + s;
            }
            sb.append(s);
        }
        return sb.toString();
    }

    /**
     * 求两个字符串的异或结果
     * 注意：两字符串长度必须相同
     */
    public static String xor(String s1, String s2) {
        if (s1 == null || s2 == null || s1.length() != s2.length())
            throw new RuntimeException("字符串长度不相等");
        StringBuilder sb = new StringBuilder(s1.length());
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) == s2.charAt(i)) {
                sb.append("0");
            } else {
                sb.append("1");
```

```
                }
            }
            return sb.toString();
        }

}
```

2. AES

```
package AESshiyan;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class AES {
    public static final String ZEROS = "00000000";
    public static final String XOR_STR = "00011011";     // 84310

    public static final String[][] FIX_COLUMN = new String[][] {     // 列混淆变
换
            {"02", "03", "01", "01"},
            {"01", "02", "03", "01"},
            {"01", "01", "02", "03"},
            {"03", "01", "01", "02"}
    };

    public static final String[][] FIX_COLUMN_INVERSE = new String[][]
{    // 列混淆逆向变换
            {"0E", "0B", "0D", "09"},
            {"09", "0E", "0B", "0D"},
            {"0D", "09", "0E", "0B"},
            {"0B", "0D", "09", "0E"}
    };

    public static final String[] RC = new String[]{"01", "02", "04", "08", "10",
"20", "40", "80", "1B", "36"};

    public static final String[][] S = new String[][]{
            {"63", "7c", "77", "7b", "f2", "6b", "6f", "c5", "30", "01", "67",
"2b", "fe", "d7", "ab", "76"},
            {"ca", "82", "c9", "7d", "fa", "59", "47", "f0", "ad", "d4", "a2",
"af", "9c", "a4", "72", "c0"},
            {"b7", "fd", "93", "26", "36", "3f", "f7", "cc", "34", "a5", "e5",
"f1", "71", "d8", "31", "15"},
            {"04", "c7", "23", "c3", "18", "96", "05", "9a", "07", "12",
"80", "e2", "eb", "27", "b2", "75"},
```

```
                {"09", "83", "2c", "1a", "1b", "6e", "5a", "a0", "52", "3b",
"d6", "b3", "29", "e3", "2f", "84"},
                {"53", "d1", "00", "ed", "20", "fc", "b1", "5b", "6a", "cb",
"be", "39", "4a", "4c", "58", "cf"},
                {"d0", "ef", "aa", "fb", "43", "4d", "33", "85", "45", "f9", "02",
"7f", "50", "3c", "9f", "a8"},
                {"51", "a3", "40", "8f", "92", "9d", "38", "f5", "bc", "b6",
"da", "21", "10", "ff", "f3", "d2"},
                {"cd", "0c", "13", "ec", "5f", "97", "44", "17", "c4", "a7", "7e",
"3d", "64", "5d", "19", "73"},
                {"60", "81", "4f", "dc", "22", "2a", "90", "88", "46", "ee", "b8",
"14", "de", "5e", "0b", "db"},
                {"e0", "32", "3a", "0a", "49", "06", "24", "5c", "c2", "d3",
"ac", "62", "91", "95", "e4", "79"},
                {"e7", "c8", "37", "6d", "8d", "d5", "4e", "a9", "6c", "56", "f4",
"ea", "65", "7a", "ae", "08"},
                {"ba", "78", "25", "2e", "1c", "a6", "b4", "c6", "e8", "dd",
"74", "1f", "4b", "bd", "8b", "8a"},
                {"70", "3e", "b5", "66", "48", "03", "f6", "0e", "61", "35", "57",
"b9", "86", "c1", "1d", "9e"},
                {"e1", "f8", "98", "11", "69", "d9", "8e", "94", "9b", "1e", "87",
"e9", "ce", "55", "28", "df"},
                {"8c", "a1", "89", "0d", "bf", "e6", "42", "68", "41", "99",
"2d", "0f", "b0", "54", "bb", "16"}

        };

        public static final String[][] S_INVERSE = new String[][]{
                {"52", "09", "6a", "d5", "30", "36", "a5", "38", "bf", "40",
"a3", "9e", "81", "f3", "d7", "fb"},
                {"7c", "e3", "39", "82", "9b", "2f", "ff", "87", "34", "8e", "43",
"44", "c4", "de", "e9", "cb"},
                {"54", "7b", "94", "32", "a6", "c2", "23", "3d", "ee", "4c",
"95", "0b", "42", "fa", "c3", "4e"},
                {"08", "2e", "a1", "66", "28", "d9", "24", "b2", "76", "5b",
"a2", "49", "6d", "8b", "d1", "25"},
                {"72", "f8", "f6", "64", "86", "68", "98", "16", "d4", "a4", "5c",
"cc", "5d", "65", "b6", "92"},
                {"6c", "70", "48", "50", "fd", "ed", "b9", "da", "5e", "15",
"46", "57", "a7", "8d", "9d", "84"},
                {"90", "d8", "ab", "00", "8c", "bc", "d3", "0a", "f7", "e4",
"58", "05", "b8", "b3", "45", "06"},
                {"d0", "2c", "1e", "8f", "ca", "3f", "0f", "02", "c1", "af", "bd",
"03", "01", "13", "8a", "6b"},
                {"3a", "91", "11", "41", "4f", "67", "dc", "ea", "97", "f2", "cf",
"ce", "f0", "b4", "e6", "73"},
```

```
            {"96", "ac", "74", "22", "e7", "ad", "35", "85", "e2", "f9", "37",
"e8", "1c", "75", "df", "6e"},
            {"47", "f1", "1a", "71", "1d", "29", "c5", "89", "6f", "b7", "62",
"0e", "aa", "18", "be", "1b"},
            {"fc", "56", "3e", "4b", "c6", "d2", "79", "20", "9a", "db",
"c0", "fe", "78", "cd", "5a", "f4"},
            {"1f", "dd", "a8", "33", "88", "07", "c7", "31", "b1", "12",
"10", "59", "27", "80", "ec", "5f"},
            {"60", "51", "7f", "a9", "19", "b5", "4a", "0d", "2d", "e5",
"7a", "9f", "93", "c9", "9c", "ef"},
            {"a0", "e0", "3b", "4d", "ae", "2a", "f5", "b0", "c8", "eb",
"bb", "3c", "83", "53", "99", "61"},
            {"17", "2b", "04", "7e", "ba", "77", "d6", "26", "e1", "69",
"14", "63", "55", "21", "0c", "7d"}

    };


    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String choose = null;
        while (true) {
            System.out.println("AES 算法，请选择您要进行的操作(按下对应
操作数字后回车): (1)加密  (2)解密  (3)退出");
            choose = scanner.nextLine();
            if ("1".equals(choose)) {
                System.out.print("请输入明文(不含中文): ");
                String plainText = scanner.nextLine();
                System.out.print("请输入密钥(16 个字符以内，不含中文): ");
                String secret = scanner.nextLine();
                String cipher = encrypt(plainText, secret);
                System.out.println("加密结果: " + cipher);
            } else if ("2".equals(choose)) {
                System.out.print("请输入密文(不含中文): ");
                String cipher = scanner.nextLine();
                System.out.print("请输入密钥(16 个字符以内，不含中文): ");
                String secret = scanner.nextLine();
                String plainText = decrypt(cipher, secret);
                System.out.println("解密结果: " + plainText);
            } else if ("3".equals(choose)) {
                System.exit(0);
            }
        }

    }
```

```
    public static String encrypt(String plainText, String secret) {
        // 0. 明文/密钥分组
        String[][][] plainMatrix = divide4PlainText(plainText);
        String[][] initSecret = divide4Secret(secret);
        StringBuilder sb = new StringBuilder(plainMatrix.length << 4); // 用于拼
接密文
        // 1. 构造密钥组
        String[][][] extendSecret = extendSecret(initSecret);
        for (int i = 0; i < plainMatrix.length; i++) {
            // 2. 轮密钥加
            plainMatrix[i] = xor4HexArr(plainMatrix[i], extendSecret[0]);
            // 3.10 轮重复加密操作
            for (int round = 1; round <= 10; round++) {
                // 字节代替
                replaceByte(plainMatrix[i], S);
                // 行移位
                plainMatrix[i] = moveRow(plainMatrix[i]);
                // 列混淆(第十轮不进行)
                if (round != 10) {
                    plainMatrix[i] = fixColumn(plainMatrix[i], FIX_COLUMN);
                }
                // 轮密钥加
                plainMatrix[i] = xor4HexArr(plainMatrix[i],
extendSecret[round]);
            }
            // 4.拼接密文
            for (int k = 0; k < 4; k++) {
                for (int j = 0; j < 4; j++) {
                    sb.append(plainMatrix[i][j][k]);
                }
            }
        }
        return sb.toString();
    }

    /**
     * 解密
     * @param cipher 密文 16 进制字符串
     * @param secret 密钥字符串
     */
    public static String decrypt(String cipher, String secret) {
        // 0. 密文/密钥分组
        String[][][] cipherMatrix = divide4Hex(cipher);
        String[][] initSecret = divide4Secret(secret);
        List<Byte> list = new ArrayList<>(cipherMatrix.length << 4 + 16);
```

```
            // 1. 构造密钥组
            String[][][] extendSecret = extendSecret(initSecret);
            for (int i = 0; i < cipherMatrix.length; i++) {
                // 2. 轮密钥加
                cipherMatrix[i] = xor4HexArr(cipherMatrix[i], extendSecret[10]);
                // 3.10 轮重复加密操作
                for (int round = 9; round >= 0; round--) {
                    // 逆行移位
                    cipherMatrix[i] = moveRowInverse(cipherMatrix[i]);
                    // 逆字节代替
                    replaceByte(cipherMatrix[i], S_INVERSE);
                    // 轮密钥加
                    cipherMatrix[i] = xor4HexArr(cipherMatrix[i],
extendSecret[round]);
                    // 逆列混淆(第十轮不进行)
                    if (round != 0) {
                        cipherMatrix[i] = fixColumn(cipherMatrix[i],
FIX_COLUMN_INVERSE);
                    }
                }
                // 4.还原明文
                for (int k = 0; k < 4; k++) {
                    for (int j = 0; j < 4; j++) {
                        byte byteVal = Byte.valueOf(cipherMatrix[i][j][k], 16);
                        if (byteVal != 0) {
                            list.add(byteVal);
                        }
                    }
                }
            }
            byte[] bytes = new byte[list.size()];
            for (int i = 0; i < bytes.length; i++) {
                bytes[i] = list.get(i);
            }
            return new String(bytes);
        }


        /**
         * 打印 4 * 4 矩阵
         */
        public static void print(String[][] arr) {
            for (int row = 0; row < 4; row++) {
                for (int col = 0; col < 4; col++) {
                    System.out.print(arr[row][col] + " ");
                }
```

```
                System.out.println();
            }
            System.out.println("==============");
        }

        /**
         * 明文
         * 1. 将明文的普通字符串转化成对应的多个字节数组
         * 2. 每个字节数组都不能超过 16 字节
         * 3. 将字节数组按列排成 4 * 4 矩阵
         * 4. 字节不足时用 00 充当
         */
        public static String[][][] divide4PlainText(String plainText) {
            byte[] bytes = plainText.getBytes();
            int len = (bytes.length & 15) == 0    ? (bytes.length >>> 4) :
(bytes.length >>> 4) + 1;
            String[][][] ret = new String[len][4][4];
            for (int k = 0; k < len; k++) {
                for (int i = 0; i < 4; i++) {
                    for (int j = 0; j < 4; j++) {
                        int index = (i << 2) + j + (k << 4);
                        if (index < bytes.length) {
                            String hex = Integer.toHexString(bytes[index]);
                            if (hex.length() == 1) {
                                hex = "0" + hex;
                            }
                            ret[k][j][i] = hex;
                        } else {
                            ret[k][j][i] = "00";
                        }
                    }
                }
            }
            return ret;
        }
        /**
         * 密钥分组
         * 1. 密钥的普通字符串转化成对应的字节数组
         * 2. 不能超过 16 字节
         * 3. 将字节数组按列排成 4 * 4 矩阵
         * 4. 字节不足时用 00 充当
         */
        public static String[][] divide4Secret(String secret) {
            String[][] ret = new String[4][4];
            byte[] bytes = secret.getBytes();
            if (bytes.length > 16) {
```

```
            throw new RuntimeException("密钥不能超过 16 个字节");
        }
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                int index = (i << 2) + j;
                if (index < bytes.length) {
                    String hex = Integer.toHexString(bytes[index]);
                    if (hex.length() == 1) {
                        hex = "0" + hex;
                    }
                    ret[j][i] = hex;
                } else {
                    ret[j][i] = "00";

                }
            }
        }
        return ret;
    }


    /*
     *
     * 将十六进制字符串分成多组 4 * 4
     * 如: 0123456789abcdeffedcba9876543210    (16 字节)
     * 转成:
                01 89 fe 76
               23 ab dc 54
               45 cd ba 32
               67 ef 98 10
     * 转入的是密文(长度一定是 32 的倍数)
     */
    public static String[][][] divide4Hex(String hexString) {
        int len = hexString.length() >>> 5;
        String[][][] ret = new String[len][4][4];
        int index = 0;
        for (int k = 0; k < len; k++) {
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    ret[k][j][i] = hexString.substring(index, index + 2);
                    index += 2;
                }
            }
        }
        return ret;
    }
```

```java
    /**
     * 密钥扩展
     */
    public static String[][][] extendSecret(String[][] initSecret) {

        String[][] w = new String[44][4];
        // 1. w0-w3 等于初始密钥
        for (int i = 0; i < initSecret.length; i++) {
            for (int j = 0; j < 4; j++) {
                w[i][j] = initSecret[j][i];
            }
        }
        // 2.计算 w4-w43
        for (int i = 4; i < 44; i++) {
            String[] temp = w[i - 1];
            if ((i & 3) == 0) { // 如果 wi 下标 i 可以被 4 整除，则 w[i] = w[i - 4] xor z[i >>> 2]
                temp =    g(w[i - 1], (i >>> 2) - 1);
            } // 否则, w[i] = w[i - 1] xor w[i - 4]
            for (int j = 0; j < 4; j++) {
                w[i][j] = xor4Hex(w[i - 4][j], temp[j]);
            }
        }

        // 3.计算构造密钥组返回
        String[][][] secretGroup = new String[11][4][4];
        for (int i = 0; i < 11; i++) {
            for (int j = 0; j < 4; j++) {
                for (int k = 0; k < 4; k++) {
                    secretGroup[i][j][k] = w[(i << 2) + k][j];
                }
            }
        }
        return secretGroup;
    }


    /**
     * 辅助函数 g
     * @param w 当前轮最后一列
     * @param round 当前轮数
     * @return
     */
    public static String[] g(String[] w, int round) {
        String[] z = new String[4];
```

```
        for (int i = 0; i < 4; i++) {
            // 1.字循环
            // 2.字代替
            z[i] = searchS(w[i + 1 & 3], S);
            // 3.xor 轮常数
            if (i == 0) {
                z[i] = xor4Hex(z[i], RC[round]);
            }
        }
        return z;
    }

    /**
     * 字节替换
     */
    public static void replaceByte(String[][] arr, String[][] s) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                arr[i][j] = searchS(arr[i][j], s);
            }
        }
    }

    /**
     * 行移位
     */
    public static String[][] moveRow(String[][] arr) {
        String[][] newArr = new String[4][4];
        for (int row = 0; row < 4; row++) {
            for (int col = 0; col < 4; col++) {
                newArr[row][col] = arr[row][(col + row) & 3];
            }
        }
        return newArr;
    }

    /**
     * 逆行移位
     */
    public static String[][] moveRowInverse(String[][] arr) {
        String[][] newArr = new String[4][4];
        for (int row = 0; row < 4; row++) {
            for (int col = 0; col < 4; col++) {
                newArr[row][col] = arr[row][(4 - row + col) & 3];
            }
        }
```

```
        return newArr;
    }

    /**
     * 列混淆
     */
    public static String[][] fixColumn(String[][] source, String[][] fix) {
        String[][] result = new String[4][4];
        for (int i = 0; i < 4; i++) {   // 控制行
            for (int j = 0; j < 4; j++) {   // 控制行使用次数
                String ret = "00000000";
                for (int k = 0; k < 4; k++) {   // 控制列
                    String binMultiply = binMultiply(fix[i][k], source[k][j]);
                    ret = xor(ret, binMultiply);
                }
                // 将结果转为 16 进制
                String hex = binToHex(ret);
                result[i][j] = hex;
            }
        }
        return result;
    }

    /**
     * S 盒替换
     * @return
     */
    public static String searchS(String hex, String[][] s) {
        /*
         * 1. 将传入的 2 位 16 进制字符串前后位分开，转成 10 进制
         * 2. 第一位表示行，第 2 位表示列，去查询 S 盒，返回 2 位 16 进制字符串
         */
        int row = Integer.valueOf(hex.substring(0, 1), 16);
        int col = Integer.valueOf(hex.substring(1, 2), 16);
        return s[row][col];
    }

    /**
     * 实现两个 4 * 4 矩阵对应元素的 xor 操作
     * @return
     */
    public static String[][] xor4HexArr(String[][] h1, String[][] h2) {
        String[][] ret = new String[4][4];
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
```

```
                ret[i][j] = xor4Hex(h1[i][j], h2[i][j]);
            }
        }
        return ret;
    }

    /**
     * 将两个 16 进制字符串转成 2 进制字符串后进行乘法运算，结果符合 GF(2^8)
     */
    public static String binMultiply(String hexStr1, String hexStr2) {
        // 1. 将 16 进制字串符转成二进制字符串，不足 8 位则在前面补 0
        String binStr1 = hexTobin(hexStr1);
        String binStr2 = hexTobin(hexStr2);

        // 2. 定义返回结果 result，遍历 binStr1 中字符，遇到字符'1'时，记录其索引 index，调用 binMultiply(binStr2, 7 - index)，得到 ret，再与 result 进行 xor 运算
        String result = "00000000";
        for (int i = 0; i < binStr1.length(); i++) {
            if (binStr1.charAt(i) == '1') {
                String ret = binMultiply(binStr2, 7 - i);
                result = xor(result, ret);
            }
        }
        return result;
    }

    /**
     * 计算 x 的 n 次方 *f(x)
     * @param binStr f(x)二进制字符串
     * @param num x 的指数 n
     */
    public static String binMultiply(String binStr, int num) {
        /*
            思路:
                (1)循环执行 num 次以下的操作
                (2)是否 flag 判断最左边位是否为 1
                (3)将 binStr 中二进制数左移一位，右边补 0
                (4)如果 flag 为 true，将(3)所得结果与 00011011 进行 xor 操作
         */
        String ret = new String(binStr);
        for (int i = 0; i < num; i++) {
            boolean flag = ret.charAt(0) == '1';
            ret = ret.substring(1, ret.length()) + "0";
            if (flag) {
```

```
                ret = xor(ret, XOR_STR);
            }
        }
        return ret;
    }

    /**
     * 两个十六进制字符串求异或
     */
    public static String xor4Hex(String h1, String h2) {
        return binToHex(xor(hexTobin(h1), hexTobin(h2)));
    }

    /**
     * 两个二进制字符串求异或
     */
    public static String xor(String s1, String s2) {
        if (s1 == null || s2 == null || s1.length() != s2.length())
            throw new RuntimeException("字符串长度不相等");
        StringBuilder sb = new StringBuilder(s1.length());
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) == s2.charAt(i)) {
                sb.append("0");
            } else {
                sb.append("1");
            }
        }
        return sb.toString();
    }

    /**
     * 8 位二进制转 2 位十六进制
     * @return
     */
    public static String binToHex(String bin) {
        String hex = Integer.toHexString(Integer.valueOf(bin, 2));
        if (hex.length() == 1) {
            hex = "0" + hex;
        }
        return hex;
    }

    /**
     * 2 位十六进制转 8 位二进制
     * @param hex
     * @return
```

```
     */
    public static String hexTobin(String hex) {
        String bin = Integer.toBinaryString(Integer.valueOf(hex, 16));
        int len1 = 8 - bin.length();
        if (len1 != 0) {
            bin = ZEROS.substring(0, len1) + bin;
        }
        return bin;
    }
}
```

3. RSA

```
package RSAashiyan;
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateCrtKey;
import java.security.interfaces.RSAPublicKey;
import java.util.logging.Level;
import java.util.logging.Logger;

public class RSA {
    /**
     * 创建密钥对生成器，指定加密和解密算法为RSA
     *
     * @param keylen
     * @return
     */
    public String[] createKey(int keylen) {// 输入密钥长度
        String[] output = new String[5]; // 用来存储密钥的 e n d p q
        try {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(keylen); // 指定密钥的长度，初始化密钥对生成器
            KeyPair kp = kpg.generateKeyPair(); // 生成密钥对
            RSAPublicKey puk = (RSAPublicKey) kp.getPublic();
            RSAPrivateCrtKey prk = (RSAPrivateCrtKey) kp.getPrivate();
            BigInteger e = puk.getPublicExponent();
            BigInteger n = puk.getModulus();
            BigInteger d = prk.getPrivateExponent();
            BigInteger p = prk.getPrimeP();
            BigInteger q = prk.getPrimeQ();
            output[0] = e.toString();
```

```
            output[1] = n.toString();
            output[2] = d.toString();
            output[3] = p.toString();
            output[4] = q.toString();
        } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null,
ex);
        }
        //System.out.println(output[0]);
        return output;
    }


    /**
     * 加密在RSA 公钥中包含有两个整数信息：e 和n。对于明文数字m,计算
密文的公式是m 的e 次方再与n 求模。
     *
     * @param clearText  明文
     * @param eStr       公钥
     * @param nStr
     * @return
     */
    public String encrypt(String clearText, String eStr, String nStr) {
        String secretText = "";

        try {
            clearText = URLEncoder.encode(clearText, "GBK");
            byte[] text = clearText.getBytes("GBK");//将字符串转换成byte 类
型数组，实质是各个字符的二进制形式
            BigInteger mm = new BigInteger(text);//二进制串转换为一个大整数
            clearText = mm.toString();

            BigInteger e = new BigInteger(eStr);
            BigInteger n = new BigInteger(nStr);
            byte[] ptext = clearText.getBytes(StandardCharsets.UTF_8); //  获取
明文的大整数
            BigInteger m = new BigInteger(ptext);
            BigInteger c = m.modPow(e, n);
            secretText = c.toString();
        } catch (UnsupportedEncodingException ex) {
            Logger.getLogger(RSA.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return secretText;
    }
```

```
/**
 * 解密
 *
 * @param secretText  密文
 * @param dStr        私钥
 * @param nStr
 * @return
 */
public String decrypt(String secretText, String dStr, String nStr) {
    StringBuilder clearTextBuffer = new StringBuilder();
    BigInteger d = new BigInteger(dStr);// 获取私钥的参数 d,n
    BigInteger n = new BigInteger(nStr);
    BigInteger c = new BigInteger(secretText);
    BigInteger m = c.modPow(d, n);// 解密明文
    byte[] mt = m.toByteArray();// 计算明文对应的字符串并输出
    for (int i = 0; i < mt.length; i++) {
        clearTextBuffer.append((char) mt[i]);
    }
    String temp = clearTextBuffer.toString();//temp 为明文的字符串形式
    BigInteger b = new BigInteger(temp);//b 为明文的 BigInteger 类型
    byte[] mt1 = b.toByteArray();

    try {
        String clearText = (new String(mt1, "GBK"));
        clearText = URLDecoder.decode(clearText, "GBK");
        return clearText;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }

    return null;
}

public static void main(String[] args){
    RSA rsa = new RSA();
    String[] str = rsa.createKey(1024);
    for (String key: str) {
        System.out.println("key : \n" + key);
    }
    System.out.println(str.length);
    String clearText = "123abc 测试";
    String secretText = rsa.encrypt(clearText,  str[0], str[1]);
    System.out.println("明文是： " + clearText);
    System.out.println("密文是： " + secretText);
    System.out.println("解密后的明文是： " + rsa.decrypt(secretText, str[2],
```

```
str[1]));

        }
}
```